doi:10.5281/zenodo.17538015

New Lower Bounds for Snake-in-the-Box in 11-, 12-, and 13-dimensional Hypercubes

Thomas E. Ace
tom@minortriad.com

Abstract. The Snake-in-the-Box problem is the challenge of finding the longest possible induced path in the edge graph of an n-dimensional hypercube. Although the problem is unsolved in hypercubes of dimension 9 and above, research continues to refine lower and upper bounds on maximum possible path length. This paper presents new lower bounds of 732, 1439, and 2854 in 11, 12, and 13 dimensions, respectively, and describes the simple heuristics used in their discovery.

1 Introduction

The n-dimensional hypercube graph Q_n is a regular graph with 2^n vertices and $n{\cdot}2^{n-1}$ edges. The Snake-in-the-Box problem seeks to find the longest induced path in Q_n. No two non-consecutive vertices in a snake's path may be adjacent to each other on the hypercube, a constraint that is essential to the character and complexity of the problem.

Figure 1 shows a 3-dimensional hypercube (i.e., a cube) with unit-length edges and one vertex at the origin. Vertices are labeled with binary numbers `000` through `111` formed from Cartesian coordinates. The red path shows a length-4 snake with vertex sequence `000,001,011,111,110`.

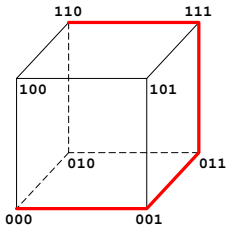

Fig. 1. A snake in 3 dimensions.

Numbering vertices in binary allows for concise expression of the constraints in the Snake-in-the-Box problem: labels of consecutive vertices in a snake's path differ in exactly one bit position (i.e., have Hamming distance 1) whereas labels

of non-consecutive vertices differ in multiple bit positions (Hamming distance > 1). The Snake-in-the-Box problem arose in the study of coding theory [3] and continues to have application in that field. Thomas Drapela's primer [2] reviews the history of the problem.

2 Search tree

Transition sequence notation for snakes is a frequently-used alternative to vertex sequence notation. In a transition sequence, each edge of a snake is labeled with the position of the bit that varies between the labels for the two vertices incident to that edge, i.e. the base 2 logarithm of the bitwise exclusive OR of the vertices' binary labels. By convention, the initial vertex is at the origin. Transition sequence notation for the snake in Figure 1 is thus `0,1,2,0`. Commas are often omitted in transition sequences.

The set of transition sequences for all possible snakes starting from the origin of a hypercube can be organized in a tree whose root node is a degenerate (empty transition sequence) snake. Transition sequences for each node's children are formed by appending digits corresponding to growth of the path in directions that create legal snakes.

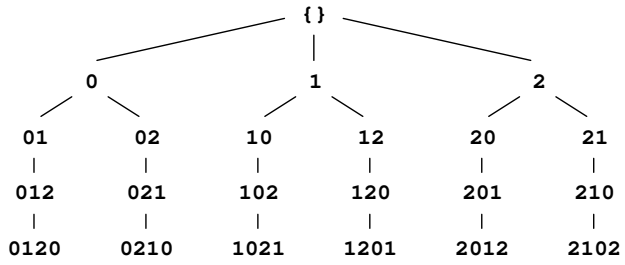

Fig. 2. The complete tree of snakes in 3 dimensions.

Figure 2 shows the tree for the cube as shown in Figure 1. The first step in a path from the origin can traverse an edge in any of the three spatial dimensions. The second step can continue in either of two directions. There is only one legal choice of direction for the third and fourth edges added. Thereafter no further extension of a snake is possible. The tree for 3 dimensions is small enough to be exhaustively searched at a glance, revealing that the longest possible snake in a cube has 4 edges.

The trend of decreasing branching factor in successive levels of the tree is characteristic of the Snake-in-the-Box problem. Options for continued growth of a snake dwindle as the snake grows to occupy more vertices of the hypercube.

Figure 2 shows that there are six possible length-4 snakes starting from the origin of a cube. Their shapes are however not unique; they are all the same snake as transformed by the symmetries of a cube. Naïvely searching the entire

tree is clearly wasteful. It is more efficient to prune the tree to contain only one snake from each equivalence class of intertransformable snakes. To that end, Kochut [5] described a canonical form of transition sequence that requires new directions to be introduced in ascending order when extending a snake. Thus:

- the first digit in a transition sequence must be zero
- each subsequent digit must either be one of digits preceding it or one more than the maximum preceding digit in the sequence.

Following those rules, there is only one choice available at each level of the tree of snakes in 3 dimensions. The tree in Figure 2 prunes down to the single path on its left edge leading to leaf node `0120`. There is no loss of generality, i.e. a tree pruned by this method will contain an instance of the longest possible snake in a hypercube.

Pruning by canonical form is worthwhile but it scarcely puts a dent in the combinatorial explosion that makes trees for hypercubes of dimension 9 or higher monstrously large. Extensive pruning is required to profitably search trees of snakes in high-dimensional hypercubes.

3 Heuristic Tree Search

The program which found the snakes described herein performs a heuristically-pruned breadth-first search (known as *beam search* in some literature, e.g. Meyerson [6]) of a tree of possible snakes structured as shown in Figure 2.

No more than two levels of the tree are represented in memory at any given time. After all children of a level's nodes have been generated, memory for the parent nodes is freed for future use.

Each node in the tree represents a snake and the hypercube it inhabits. Vertices of an n-dimensional hypercube are represented by 2^n bits, one bit for each vertex, initialized to all zeroes. Vertices prohibited by being adjacent to vertices added to the snake's path are marked by one bits. This leaves the core of the snake hollow—it is a chain of adjacent zero bits that can be retraced once the snake has been extended as far as possible.

The program starts pruning the tree when generating all of a level's children would exceed the amount of available memory. This is based on an estimate, as the exact branching factor for the next level cannot be known in advance.

Selecting which nodes at a given level to discard is guided by assigning to each node a measure of fitness that favors nodes deemed more likely to lead to long snakes. Pruning by such a measure is inherently imperfect and runs the risk of discarding nodes with the best future prospects.

The program discards nodes whose count of unmarked (zero-bit) vertices falls below a threshold calculated to ensure that generating all of the next-level children will not exceed memory limitations.

A variety of fitness measures can be calculated from the configuration of free and prohibited vertices in a hypercube. Meyerson [6] proposes counting unreachable vertices, dead ends, and blind alleys. Yet a simple count of remaining unmarked (zero-bit) vertices was sufficient to achieve the results presented here.

4 Results

The snakes described herein were all found by extending a long snake in the next lower dimension. Kinny [4] notes that "priming is an inherently unsafe method for finding optimal snakes. Nonetheless, priming the search with near-optimal snakes in the next lower dimension is a widely-used technique to make search tractable."

The transition sequence for the newly-found length-2854 13-dimensional snake is given in the Appendix. Truncating its transition sequence snake after 190, 373, 732, and 1439 digits gives snakes in 9, 10, 11, and 12 dimensions, respectively. The length-190 snake is from Wynn [8] and the successive extensions to higher-dimensional snakes were found by the method described herein.

The best previous records known to the author in dimensions 10-13 are from Dang et al. [1], notable for also being extensions of the length-190 snake found by Wynn [8]. Whereas Dang et al. used Meta-NRPA (Nested Rollout Policy Adaptation, a form of Monte Carlo search), the method presented herein has no stochastic component. Table 1 compares results from pruned breadth-first search as described herein to those from Meta-NRPA.

dimension	**pruned BFS**	**Meta-NRPA**
7	50	50
8	97	97
9	188	188
10	373	373
11	732	721
12	1439	1383
13	2854	2709

Table 1. Comparison of pruned breadth-first-search and Meta- Nested Rollout Policy Adaptation. [1]

Continued reliance on Wynn's length-190 snake for priming suggest that his method of concatenating permuted snakes [8] merits further application and has the potential to improve on the results presented here.

Code for this work was written in C++, compiled by gcc, and run under Ubuntu Linux. Running in 10 parallel threads on an Intel i5-12600K processor and using 18GB of memory, the program takes 50 minutes to extend the length-190 snake with 183 additional edges, giving a snake of length 373 in 10 dimensions. Extending the length-1439 snake in 12 dimensions to length 2854 in 13 dimensions takes two hours and uses 19GB of memory.

The transition sequence for a length-2854 snake is given in the Appendix as part of a program which verifies its correctness.

5 Summary

Breadth-first search with simple pruning heuristics achieves potentially new record results for Snake-in-the-Box in dimensions 11 through 13. As it falls short of matching the best known results in dimensions 8 and 9 (98 and 190, respectively), there most likely remains considerable room for improvement in the results it achieved in dimensions 10-13.

6 Acknowledgements

Thanks to Randall Munroe, whose comic xkcd #3125 [7] introduced the author to the Snake-in-the-Box problem.

Thanks to everyone who has written about Snake-in-the-Box and especially to Ed Wynn, whose outstanding length-190 snake in 9 dimensions [8] provided the seed for the snakes described in this paper.

References

1. Dang, Chen; Bazgan, Cristina; Cazenave, Tristan; Chopin, Morgan; Wuillemin, Pierre-Henri (2023), Warm-Starting Nested Rollout Policy Adaptation with Optimal Stopping, *Proceedings of the AAAI Conference on Artificial Intelligence* 37 (10): 12381–12389.
2. T. E. Drapela, The Snake-in-the-Box Problem: a primer, MSc Thesis, Athens, Georgia (2015).
3. W. H. Kautz, Unit-distance error-checking codes, *IRE Trans. Electronic Computers* EC-7 (1958) 179–180.
4. D. Kinny, A new approach to the Snake-In-The-Box-problem, *Proc. ECAI* 2012, 462–467.
5. K. J. Kochut, Snake-In-The-Box codes for dimension 7, *Journal of Combinatorial Mathematics and Combinatorial Computing* 20 (1996) 175–185.
6. S. J, Meyerson, Finding Longest Paths in Hypercubes: 11 New Lower Bounds for Snakes, Coils, and Symmetrical Coils, MSc Thesis, Athens, Georgia (2015).
7. R. Munroe, Snake-in-the-Box Problem, https://xkcd.com/3125/ (2025).
8. E. Wynn, Constructing circuit codes by permuting initial sequences, arXiv:1201.1647v1 (2012).

A Appendix

The following C program validates a recently-discovered snake of length 2854 in 13 dimensions and thereby also validates the 11- and 12-dimensional snakes given by truncating the transition sequence at lengths 732 and 1439, respectively.

```
static const char  transitions[] =    /* T. Ace  5 November 2025 */
 "012031402105410214321026431450413421043145014327312014301263214 30"
 "531023053145036043140214310468061043145014310630214320350432031 45"
 "365403140543753140214310451341021431650453145041204501430540,9804"
 "503410540214054135405613412014315401341204135734504130456354130 23"
 "413610431450143106314302341380540651341201431540120314570213421 05"
 "410341204165410214307350412013503413705402103140 2,a421405413540 56"
 "134120143154012057345041304563541302340530234120360134105413401 60"
 "864013412041340630541350320135034123621034102137234105413401246 314"
 "054134635413023405302341329105402305413503204502361341201431540 13"
 "412041357345041304563541302340541360134105413401651234138532045 01"
 "430540230536543140352054013405410213723410541340160453021432035 04"
 "320314536540314054 1,ba145041204502a132015401205734504130456354130"
 "234053023412036013410541340160864013412041340630541350320135034 12"
 "362103410213723410541340124314053142630214320350432031 43a45314504"
 "120450236134120143154013412041357345041304563541302341361041450 1"
 "431063143023413805406304134563504120135034136054013405410213712 01"
 "540160523541302340541021432931054021405413540561341201431540134 12"
 "041357345041304563541304534120360134105413401608640134120413465 34"
 "135032013503412362103410213723410541340124314054134635021431053 02"
 "1435a3054135032013503412361341201431540134120413573450413045635 41"
 "302340530234120360134105413401628045314504120135034136054012031 40"
 "210541021430735041204501430540120450632014504310452354134930413 45"
 "241053140610,cb5410214321054012057345041304563541302340530234120 3"
 "601341054134016086401341204134063054135032013503412362103410213 72"
 "341054134012431405314263021432035043203143a45314504120450236134 12"
 "014315401341204135734504130456354130234136104314501431063143023 41"
 "380540630413456350412013503413605401340541021372102314504120134 01"
 "605235413023405410214329310540214054135405613412014315401341204 13"
 "573450413045635413045341203601341054134016086401341204134653413 50"
 "320135034123621034102137234105413401243140541346350214310530214 35"
 "a3054135032013503412361341201431540134120413573450413045635413023"
 "405302341203601341054134016280453145041201350341360540120314021 05"
 "410214307350412045014305401204506320145043104523541349354135032 01"
 "3503410ba5402140541354056134120143154013412041357345041304563541 3"
 "023405302341203601341054134016086401341204134653413503201350341 23"
 "621034102137234105413401243140541346350214310530214 35a30541350320"
 "135034132613412014315401341204135734504130456354130234053023412 03"
 "601341054134016280453145041201356310214305431402143104506327504 13"
 "056210650461234130456125943102140541354056134120143154013412041 35"
 "734504130456354130234053023412036013410541340160864013412041340 63"
 "054135032013503412362103410213723410541340124314054134635021431 05"
 "3143023413a46134120143154013412041357345041304563541302340530234 1"
 "203601341054134016710450341054021405380341201450120413021045076 05"
 "401340541024054134504620531021405413540672104503410540214054135";
```

```
#include <stdio.h>

int main(int argc,const char **argv)
{
   const char  *ch;
   int   d = 11,digit,i,j,k,len = 0,one_count,v[3000];
   v[0] = 0;
   printf("vertices:  %x",v[0]);
   for (ch = transitions; *ch; ch++) {
      digit = 15;
      while (digit >= 0 && "0123456789abcdef"[digit] != *ch) digit--;
      if (digit >= 0) {   /* ignore commas */
         if (digit >= d) {
            printf("\n  truncate here for d=%d len=%d\n",d++,len);
         }
         if (++len >= 3000) {
            printf(" ... snake too long\n");
            return 1;
         }
         v[len] = v[len - 1] ^ (1 << digit);
         printf("-%x",v[len]);
      }
   }
   printf("\ncomplete snake:  d=%d  len=%d\n",d,len);
   for    (i = 2; i <= len  ; i++) {
      for (j = 0; j <= i - 2; j++) {
         one_count = 0;
         for (k = v[i] ^ v[j]; k; k >>= 1) one_count += k & 1;
         if (one_count <= 1) {
            printf("violation detected between vertices "
                   "%d (0x%x) and %d (0x%x)\n",i,v[i],j,v[j]);
            return 1;
         }
      }
   }
   printf("valid snake; no violations detected\n");
   return 0;
}
```